Efficient Geometrically Exact Continuous Collision Detection

Tyson Brochu[*] University of British Columbia

Essex Edwards[*] University of British Columbia

Robert Bridson[*] University of British Columbia

Abstract

Continuous collision detection (CCD) between deforming triangle mesh elements in 3D is a critical tool for many applications. The standard method involving a cubic polynomial solver is vulnerable to rounding error, requiring the use of *ad hoc* tolerances, and nevertheless is particularly fragile in (near-)planar cases. Even with per-simulation tuning, it may still cause problems by missing collisions or erroneously flagging non-collisions. We present a **geometrically exact** alternative guaranteed to produce the correct Boolean result (significant collision or not) as if calculated with exact arithmetic, even in degenerate scenarios. Our critical insight is that only the parity of the number of collisions is needed for robust simulation, and this parity can be calculated with simpler non-constructive predicates. In essence we analyze the roots of the nonlinear system of equations defining CCD through careful consideration of the boundary of the parameter domain. The use of new conservative culling and interval filters allows typical simulations to run as fast as with the non-robust version, but without need for tuning or worries about failure cases even in geometrically degenerate scenarios. We demonstrate the effectiveness of geometrically exact detection with a novel adaptive cloth simulation, the first to guarantee to remain intersection-free despite frequent curvature-driven remeshing.

CR Categories: Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation; Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Physically based modeling

Keywords: collision detection, computational geometry, cloth simulation, physically based animation

Links: DL PDF

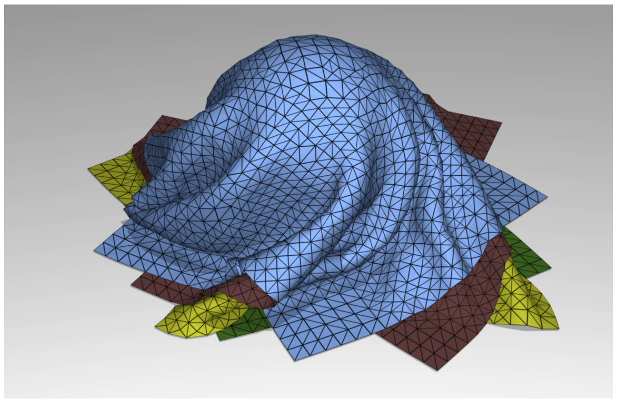

Figure 1: *Four layers of cloth folding over a spinning ball, with on-the-fly adaptive remeshing driven by curvature, provide an example where geometrically exact continuous collision detection has advantages over previous techniques.*

1 Introduction

We consider continuous collision detection (CCD) as the process of detecting if a mesh moving between initial and final configurations comes into contact with itself at any point in time. CCD is a critical element of many algorithms for physical simulation, when a non-intersecting mesh invariant must be maintained, and for path planning, when the feasibility of a path must be guaranteed. Beyond efficiency, a good CCD algorithm should therefore be *safe*: no false negatives, i.e. missed collisions, can be tolerated. It should also be *accurate* in the sense of minimizing the number of false positives, i.e. non-collisions being flagged as collisions, for the effectiveness and efficiency of algorithms using CCD. This paper provides **(1)** the first CCD algorithm to guarantee safety and accuracy despite using rounded floating-point arithmetic, under the paradigm of *Exact Geometric Computation* described by Yap [2004]: we compute the correct Boolean answer (collision or not) as if exact arithmetic were used. As part of our CCD algorithm we also present **(2)** a new, efficient, geometrically exact ray vs. bilinear patch intersection parity test, which can be used to precisely determine if a point is inside a quad-mesh-bounded volume or not. We also introduce **(3)** a new adaptive cloth simulation simulation method which maintains intersection-free meshes despite remeshing, as an example of the practical advantage of geometrically exact CCD, and show there is no performance penalty for using geometrically exact CCD.

We restrict our attention to triangle meshes in 3D, with an intersection-free initial configuration, so CCD can be reduced to two primitive tests: does a moving point hit a moving triangle, or does a moving edge hit another moving edge? Note that we ignore the connectedness of the mesh: multiple meshes are treated as lumped together, so there is no difference between inter-object collision and self-collision. We further assume that vertices move with constant velocity during the time step and that triangles are linearly interpolated between their vertices at intermediate times. If a collision does happen, we do not require its precise time and location: simple approximations detailed below are quite adequate for collision resolution.

[*](tbrochu|essex|rbridson)@cs.ubc.ca

2 Related Work

2.1 The Cubic Solver Approach

The most popular current method for continuous collision detection for triangle meshes was introduced by Provot [1997]. First a cubic equation is solved to determine coplanarity times, then the interpolated geometry is checked for overlap at these times to determine if a collision actually occurs. Bridson et al. [2002] significantly reduced the number of false negatives due to floating-point error by introducing error tolerances in the root-finding algorithm used to solve the cubic equation and using a static distance query at the coplanarity times: collisions are reported if the mesh elements are within some small distance of each other.

However, the minimum error tolerances required for safe CCD are difficult to predict in advance. Especially in cases where the primitives remain nearly coplanar for the entire step, such as hair segments [Selle et al. 2008] sliding against each other on skin, cancellation error in simply computing the coefficients of the cubic can eliminate almost all precision in the rest of the calculation. (Of course, in constantly coplanar cases, the method breaks down entirely.) Even if the cubic is represented exactly, its roots are in general irrational and must be rounded to floating-point numbers. Completing the error analysis with further bounds on the construction of the intermediate geometry at the rounded coplanarity time, bounds on the calculated barycentric coordinates of closest points, and then bounds on the distance appears intractable. In practice, a usable tolerance can typically be found by trial and error for a large class of similar simulations (e.g. cloth animations), but different applications such as adaptive cloth, hair, or liquid surface tracking can require enervating per-simulation adjustment, which makes writing a general purpose library especially tricky.

The cubic approach naturally gives false positives if the tolerance is high enough to work. If the tolerance is too high (a definite possibility if restricted to single precision arithmetic, for example) this can seriously slow down or even completely stymie collision resolution: tuning the tolerance for a new simulation isn't always easy.

A fully symbolic implementation could in principle resolve the above problems, apart from the degenerate constantly coplanar case, but the computational overhead would be drastic. In this paper we show a different approach to CCD can be fully safe and accurate without need for tuning, yet run just as fast.

2.2 Other Work in Continuous Collision Detection

Stam [2009] extended the cubic solver approach to explicitly test if two mesh elements approach closer than a given distance during the time step, resulting in a sixth degree polynomial to solve for potential collision times. This helps to resolve the coplanar-motion degeneracy mentioned above, but poses an even less tractable rounding error analysis problem for safe CCD, suffers from the same false-positive issues, and is a heavier burden computationally.

Alternative methods for computing the time of possible collisions, such as conservative local advancement [Tang et al. 2010a] offer potential speed-ups over the cubic solver approach, but don't robustly deal with rounding error, relying on user-set tolerances to account for slight non-planarities in intersection/proximity testing.

The constant vertex velocity model underlying this paper and the cubic solver approach is perhaps the most natural for general deformable motions. However, for rigid bodies, constant linear and angular velocity of the entire model makes more sense — though the helical trajectories of vertices are somewhat more difficult to handle. Zhang et al. [2007] demonstrate significant acceleration of conservative advancement using the Taylor model generalization of interval arithmetic, but again rely on user-set tolerances to cope with the inexact solve and rounding error.

Brochu & Bridson [2009a; 2009b] suggest using a simplicial space-time mesh to model the motion of the mesh, reducing CCD to simplex intersection tests in four dimensions. While these tests could be computed exactly with known determinant-based predicates, this approximation leads to an unintuitive model for the mesh geometry at intermediate times: mesh edges develop kinks and triangles develop folds; normals do not vary continuously over time. This precludes the use of CCD culling techniques which assume the geometry is linearly interpolated at intermediate times [Tang et al. 2010b]. More importantly, the unusual model of motion causes unintuitive and undesired collisions. For example, two close but parallel triangles can move together with no collisions in the standard model, but their non-standard model can crease the triangles in an inconsistent way, causing a hard-to-resolve collision.

Raytracing can be seen as a special case of CCD, generally easier since the geometry is static relative to the "motion" of the light ray. We highlight Ramsey et al.'s ray-bilinear patch test [2004] as particularly relevant, as our 3D CCD test in fact relies on ray-bilinear patch intersection parity tests. However, our new approach is geometrically exact and fully robust, unlike Ramsey et al.'s constructive approach which is vulnerable to rounding error; on the other hand our test only provides the parity of the number of intersections, not their location.

The related problem of culling collision tests is very well studied in computer graphics. Several approaches using bounding volume hierarchies have been proposed, as well as culling using bounding boxes with regular grids, sweep-and-prune testing, and sweeping-plane testing: see Ericson's book for example [2004]. We observe that except for axis-aligned methods which only use comparisons (no arithmetic), the culling literature generally does not worry about verifiably handling rounding error. We briefly address this issue later, but emphasize our focus is the correctness of the core element vs. element test, not the efficiency of broader culling methods.

2.3 Exact Geometric Computation

The cubic solver approach is an example of a *constructive* geometric algorithm, in that intermediate geometric quantities are computed (such as planarity times and interpolated positions) and used in a sequence of calculations. In this, as in many other geometric tests, the necessary rounding analysis to get a provably correct algorithm (accounting for the errors in all intermediate quantities) is intractable while the symbolic or exact arithmetic version would be too slow (necessitating radicals in this case).

An alternative approach is to decompose a geometric test into a set of simpler *predicates*, providing discrete answers such as "does a point lie to the left, to the right or on a line?" rather than continuous values. Approximate continuous values may be computed alongside, of course, but the discrete correctness of the algorithm as a whole relies only on the correctness of the discrete answers from the predicates. Several approaches to defining and implementing correct predicates exist; the most successful is the paradigm of Exact Geometric Computation (EGC). We recommend Yap's article as an excellent review of the topic [2004], and the CGAL project for examples of applications and ongoing research [CGA]. In brief, a *geometrically exact* predicate must return the same discrete answer as if computed with exact arithmetic (from the floating point input) even if under the hood it takes a faster approach. Our method is the first geometrically exact CCD test for general CCD, but exact predicates for other problems have long been used in graphics and elsewhere.

Building on previous work by Dekker [1971] and Priest [1991], Shewchuk presented practical systems for exactly evaluating a number of geometric predicates needed for Delaunay mesh generation [1996]. These sign-of-determinant predicates are equally useful for detecting self-intersections for triangle meshes. When higher precision than provided by floating point hardware is required, the system uses *floating-point expansions* (the sum of a sequence of floats) leveraging fast floating-point hardware even for exact arithmetic.

In this paper we decompose CCD into a set of simplex intersection tests, based on the same standard sign-of-determinant tests, together with the evaluation of the sign of a simple polynomial function. No radicals or even divisions are required, making it straightforward to implement exactly using expansion arithmetic like Priest and Shewchuk. Furthermore, through the use of fast interval arithmetic filters, we can rapidly find the provably correct signs without need for high precision expansions in all but the most extreme cases, leading to highly efficient execution on average.

3 Continuous Collision Detection in 3D

In 3D CCD, there are two fundamental collisions tests: point-triangle and segment-segment. The input to each are the location of the vertices at the beginning and end of the time step. We denote the location in space of vertex i at the beginning of the time step as $\mathbf{x}_i$, and its location at the end of the time step as $\hat{\mathbf{x}}_i$. Then for each test, we are given 8 points: $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\hat{\mathbf{x}}_0$, $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$, and $\hat{\mathbf{x}}_3$. For convenience, we will normalize the time step so that $t \in [0,1]$. The constant velocity model of motion gives the location of a vertex at an intermediate time as $\mathbf{x}_i(t) = (1-t)\mathbf{x}_i + t\hat{\mathbf{x}}_i$.

First consider the point-triangle test. In the following we will index the moving vertex with 0, and the triangle vertices as 1, 2, and 3. Any point on the triangle can be written as: $\mathbf{x}(u,v) = (1-u-v)\mathbf{x}_1 + u\mathbf{x}_2 + v\mathbf{x}_3$, where $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ are the triangle corners, and $u, v \in [0,1]$ with $u+v \leq 1$. The vector between a point on the moving triangle defined by the coordinates (u,v) and the other vertex, at time t, can then be written as:

$$\begin{aligned}\mathbf{F}(t,u,v) &= \mathbf{x}_0(t) - \big[(1-u-v)\mathbf{x}_1(t) + u\mathbf{x}_2(t) + v\mathbf{x}_3(t)\big]\\ &= (1-t)\mathbf{x}_0 + t\hat{\mathbf{x}}_0\\ &\quad - (1-u-v)\big((1-t)\mathbf{x}_1 + t\hat{\mathbf{x}}_1\big)\\ &\quad - u\big((1-t)\mathbf{x}_2 + t\hat{\mathbf{x}}_2\big)\\ &\quad - v\big((1-t)\mathbf{x}_3 + t\hat{\mathbf{x}}_3\big).\end{aligned}$$

This is a tri-affine function which is zero precisely when the vertex lies on the triangle. The domain for the point-triangle test is therefore $\Omega = [0,1] \times \{u, v \geq 0 \mid u+v \leq 1\}$, a triangular prism.

A similar tri-affine function can be defined for the segment-segment collision test, the vector between the point at fraction $u \in [0,1]$ along one segment and the point at fraction $v \in [0,1]$ along the other, at time $t \in [0,1]$. The domain is then $[0,1]^3$, the unit cube.

CCD then amounts to discovering if such a function has a root in the domain, a point in Ω which $\mathbf{F}$ maps to $\mathbf{0}$. We make an important simplification: we report a collision if there is any zero on the domain of the boundary (i.e. at the initial or final time, or at any edge or endpoint of the geometry) or if there is an *odd* number of roots in the interior. We justify ignoring the case of a nonzero even number of interior roots by noting that an edge-triangle intersection cannot be introduced in the mesh if the total number of collisions between the edge and triangle has even parity. Likewise a vertex cannot enter or exit a closed mesh without either colliding with an edge or colliding with the triangles an odd number of times, and therefore with at least one triangle an odd number of times, so our method cannot miss essential collisions in this category either; see the supplemental material for more discussion.

3.1 Determining Root Parity

Write the image of Ω under $\mathbf{F}$ as $\mathbf{F}(\Omega) = \{\mathbf{y} \mid \mathbf{y} = \mathbf{F}(\mathbf{x}), \mathbf{x} \in \Omega\}$, and similarly $\mathbf{F}(\Gamma) = \{\mathbf{y} \mid \mathbf{y} = \mathbf{F}(\mathbf{x}), \mathbf{x} \in \Gamma\}$ for the image of the domain boundary $\Gamma = \partial\Omega$.

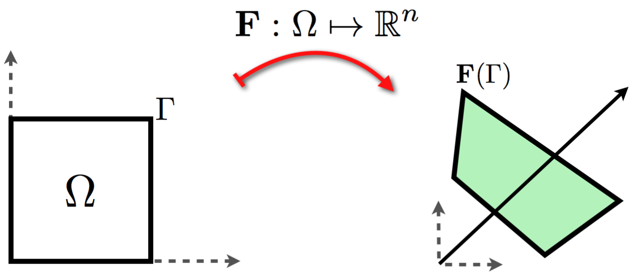

Figure 2: *Root parity test. In this case there are no roots in the domain, so the origin is outside of* $\mathbf{F}(\Omega)$.

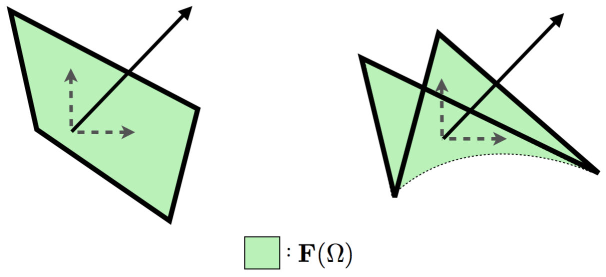

Figure 3: *One and two roots in the domain. A ray cast from the origin will have odd and even parity, respectively.*

If $\mathbf{F}$ were smooth and one-to-one, determining if $\mathbf{0} \in \mathbf{F}(\Omega)$ could be done by counting the number of crossing of a ray to infinity from $\mathbf{0}$ through the boundary image $\mathbf{F}(\Gamma)$: an odd number of crossings indicates a root by the usual Jordan-Brouwer Theorem argument. However, our $\mathbf{F}$ may not be one-to-one: the image of Ω can "fold over itself". The more general Brouwer topological degree theory [O'Regan et al. 2006] can be applied in this case. It is the parity of the ray crossings with $\mathbf{F}(\Gamma)$ that gives us the parity of the number of roots: the sum of an odd number of intersections for each separate root leads to an odd total if and only if there is an odd number of roots — with the proviso that if $\mathbf{0} \in \mathbf{F}(\Gamma)$, i.e. we have a root on the domain boundary, we always report a collision. More formally:

Root Parity Lemma. *Suppose $\Omega \subset \mathbb{R}^n$ is an n-polytope.*

Suppose $\mathbf{F} : \Omega \mapsto \mathbb{R}^n$ is C^2, has $p < \infty$ roots in Ω, has no roots on $\Gamma = \partial\Omega$, and has non-singular Jacobian at each root.

Suppose R is a ray from $\mathbf{0}$ to infinity. Call any point $\mathbf{x} \in \Gamma$ such that $\mathbf{F}(\mathbf{x}) \in R$ a crossing point, *then the* crossing number *q is the number of crossing points. Suppose that $\mathbf{F}(\Gamma)$ is smooth at the image of any crossing points, that the ray is not tangent to $\mathbf{F}(\Gamma)$ at any these points, and that $q < \infty$.*

Then, $p \equiv q \bmod 2$.

We offer a sketch of a proof of the lemma, and that it applies to the particular functions we need for CCD, in the supplemental material.

This lemma also describes our algorithm. We report a collision if the image of the boundary $\mathbf{F}(\Gamma)$ passes through $\mathbf{0}$, and otherwise cast a ray from $\mathbf{0}$ in an arbitrary direction, and then count the number of crossings of the ray though $\mathbf{F}(\Gamma)$, choosing a different direction and trying again if any crossings are tangent or lie on corners. Figures 2 and 3 illustrate this approach for the 2D case, showing cases where we have zero, one, and two roots in the domain.

We transform the boundary of these domains (cube or triangular prism) by the corresponding function $\mathbf{F}$ to get a *generalized* hexahedron or prism, and test for ray crossings on each of their faces. The hexahedron has potentially non-planar bilinear patches for faces (the restriction of the tri-affine function to a face of the domain is bi-affine), and the prism is composed of three bilinear patches and two triangles. Computing ray-triangle crossings can be done with exact arithmetic — however, we know of no prior practical method for quickly and exactly computing the crossings of a ray through a bilinear patches, or even the *parity* of the number of crossings which is all we need. We thus introduce an efficient method for exactly computing this parity.

3.2 Ray-Bilinear-Patch Crossing Parity Testing

We first define a continuous scalar function $\phi(\mathbf{x})$ which is positive if $\mathbf{x}$ is on one side of the patch and negative on the other side, and to permit exact evaluation with floating-point expansions define it using only multiplication, addition, and subtraction — see appendix A for the derivation.

Next consider the tetrahedron spanned by the four corners of the bilinear patch. It is composed of two pairs of triangles, one pair corresponding to each side of the bilinear patch. For the "positive" triangle pair, any point $\mathbf{x}$ on either triangle has the property that $\phi(\mathbf{x}) \geq 0$, and vice versa for the "negative" pair of triangles. For the test, we consider two cases depending on whether the ray origin $\mathbf{0}$ lies inside the tetrahedron or not — which can be determined directly from standard sign-of-determinant "orientation" predicates with the tetrahedron's triangular faces.

If the ray origin $\mathbf{0}$ lies inside the tetrahedron, we can determine the sign of $\phi(\mathbf{0})$ and *replace* the ray-patch test with two ray-triangle tests, using the triangles corresponding to the opposite sign of $\phi(\mathbf{0})$. Ray-triangle intersection can also be broken down into determinant predicates [Guigue and Devillers 2003]. If there is an intersection between the ray and either triangle, then the ray must also pass once through the bilinear patch.

If instead the ray origin lies outside of the tetrahedron, we can use either set of triangles as an equivalent proxy for the bilinear patch. The parity of the number of intersections between the ray and the triangle pair matches the parity of the number of intersections between the ray and the bilinear patch.

Pseudocode for the test is given in algorithm 1; figure 4 illustrates the 2D analog. Since it relies only on determinants and the evaluation of ϕ, i.e. just multiplication and addition/subtraction, the test can be evaluated using floating-point expansions to give the geometrically exact result.

Algorithm 1 Ray-bilinear-patch crossing parity

```
Given: Ray origin $\mathbf{0}$, direction $\mathbf{R}$, and a bilinear patch.
Form the tetrahedron from the bilinear patch corner vertices.
Let $F_1^+$, $F_2^+$ be the tetrahedron faces where $\phi \geq 0$.
Let $F_1^-$, $F_2^-$ be the tetrahedron faces where $\phi \leq 0$.
if $\mathbf{0}$ is inside the tetrahedron then
    if $\phi(\mathbf{0}) > 0$ then
        return intersect( $\mathbf{0}$, $\mathbf{R}$, $F_1^-$ ) $\vee$ intersect( $\mathbf{0}$, $\mathbf{R}$, $F_2^-$ )
    else
        return intersect( $\mathbf{0}$, $\mathbf{R}$, $F_1^+$ ) $\vee$ intersect( $\mathbf{0}$, $\mathbf{R}$, $F_2^+$ )
    end if
else
    {Use either pair of triangles}
    return intersect( $\mathbf{0}$, $\mathbf{R}$, $F_1^+$ ) XOR intersect( $\mathbf{0}$, $\mathbf{R}$, $F_2^+$ )
end if
```

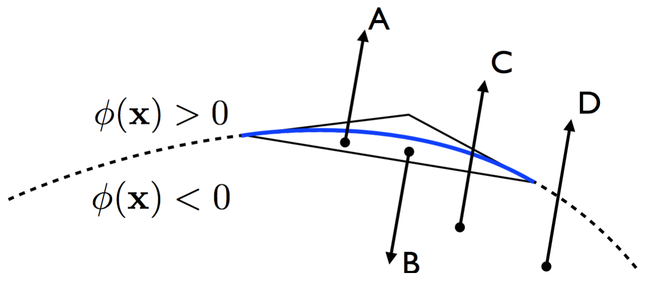

Figure 4: *A 2D analog of the ray-vs-bilinear-patch parity test. Rays A and B have origins on the "negative" side of the patch, and so we test against the proxy geometry on the "positive" side. Ray A intersects both the patch and the proxy geometry, while B intersects neither. Rays C and D have origins outside the bounding simplex, and so can be tested with either proxy geometry.*

3.3 Putting it Together

We now have the tools we need for determining the intersection parity of a ray versus a set of bilinear patches and triangles. For segment-segment CCD, our algorithm runs 6 ray-vs-patch tests for the faces of the hexahedron, and for point-triangle CCD, we run 3 ray-vs-patch and 2 ray-vs-triangle tests for the faces of the triangular prism, and determine the parity of the total number of intersections. If we have an odd parity, we know there is an odd number of roots in the domain, and so we must flag this as a collision. Algorithm 2 shows the point-vs-triangle test (the segment-vs-segment test is analogous).

There are a few special cases we must watch for. If the origin lies exactly on a patch or a triangle (i.e. there is a root on the boundary of the domain), then we report the collision and skip the ray testing. This includes, for example, the fully degenerate cases of exactly planar motion (such as two edges sliding into each other along a flat surface) that entirely defeats the cubic solver. In our simulations this type of collision is vanishingly rare unless artificially induced.

We must also take care if the ray hits an edge shared between two patches, between two triangles (acting as proxy geometry in the ray-vs-patch test), or between a patch and a triangle. If this occurs, we will see two positive ray intersection tests. In the context of inside-outside testing, this may or may not be correct (see figure 5). Fortunately, since we are using exact arithmetic, we can precisely detect these degenerate cases (one barycentric coordinate will be *exactly* zero). In such a case, we simply choose a new ray direction at random and run the test again. Again, in our testing this happens only extraordinarily rarely.

4 Implementation

Fast implementation of exact intersection testing is crucial for making our approach practical. Computing intersections with expansion arithmetic is expensive, so we use a filter: we evaluate the determinants and ϕ first with interval arithmetic, only switching to exact expansions when the sign is indeterminate (the final interval contains zero). See Brönnimann et al. for the case of determinants [2001].

To avoid repeatedly switching the rounding mode during interval arithmetic, we use the standard "opposite trick", storing the negative of the lower bound and defining new operations that rely on one rounding direction only. We have also experimented with using SIMD vectors to store the intervals and using vector intrinsics for arithmetic operations [Lambov 2006; Goualard 2010], but found

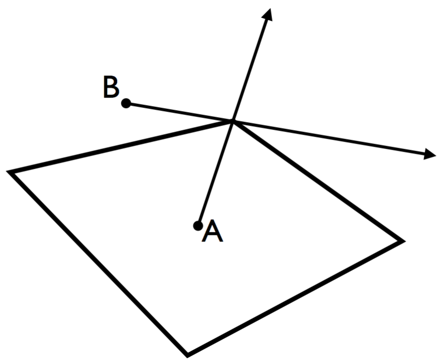

Figure 5: *Two rays, each hitting two segments at their common endpoint. If we are testing each segment individually, then in both cases the parity of ray intersections is even. Here the parity of intersection count cannot determine whether points A and B are inside the quadrilateral. Perturbing the rays slightly would produce the correct results in both cases.*

Algorithm 2 Point-triangle collision test

```
Given: corner vertices of the domain, $X$
Create ray $(\mathbf{0}, \mathbf{R})$ with arbitrary direction $\mathbf{R}$
$S \leftarrow 0$
for $i = 1 \to 3$ do
    Form bilinear patch $i$ with appropriate $\mathbf{F}(X)$
    $p_i \leftarrow$ intersection parity of ray $(\mathbf{0}, \mathbf{R})$ vs bilinear patch $i$
    $S \leftarrow S + p_i$
end for
for $j = 1 \to 2$ do
    Form triangle $j$ with appropriate $\mathbf{F}(X)$
    if Ray $(\mathbf{0}, \mathbf{R})$ intersects triangle $j$ then
        $S \leftarrow S + 1$
    end if
end for
return $S \equiv 1 \pmod 2$
```

that our implementation of this strategy was not significantly faster in practice than simply operating on two doubles.

Collision test culling is also critical for efficiency. We compute the axis-aligned bounding box (AABB) of each moving edge, triangle and vertex and only run collision detection when AABBs overlap, accelerating this test with a regular background grid. We further cull tests by checking if, for any of several non-axis normal directions, there is a plane separating the origin from the transformed hexahedron or prism. Our implementation of this plane test uses interval arithmetic for robustness: only when all vertices are definitely on the negative or positive side of a plane (no interval contains zero), do we consider the plane to be a separating plane. This relatively inexpensive plane-based testing eliminates 99% of the tests, considerably improving performance.

4.1 Resolving Collisions

We implemented our new algorithm using interval filtered floating-point expansion arithmetic, providing practical, provably robust CCD code for deforming meshes without any user-tuned tolerances. However, at this point we can only guarantee collision detection: this says nothing about *resolving* these collisions, i.e. finding a physically consistent adjustment to the final configuration of the mesh that eliminates all collisions. Indeed, taking into account that the final positions are quantized to a finite number of bits of precision, provably robust but ideally physical collision resolution may involve the solution of a rather daunting large-scale integer programming problem. As an example of the complications involved, even just transformation of an intersection-free mesh with a rotation matrix can potentially create self-intersection once the results are rounded again.

We use the velocity filtering approach initiated by Provot [1997] and extended by Bridson et al. [2002] and Harmon et al. [2008]. First repulsion forces are applied to proximal mesh elements, followed by several sweeps of CCD, applying individual impulses when collisions are detected. If there remain unresolved collisions after these sweeps, we gather overlapping collisions into "impact zones" and solve for the set of impulses which will resolve all collisions in the zone simultaneously. If this system is degenerate, we compute a single rigid motion for the vertices of the offending impact zone, ensuring no collision (modulo the quantization issue mentioned above). This was referred to as "rigid impact zones" in Bridson's original paper [2002].

While those previous works used the normal at the time of collision (typically from the triangle or from the cross-product of edges), we have found this is not a crucial choice. Interpolating the geometry at $t = 0.5$ and computing the normal from the vector between the closest points on the two mesh elements has worked equally well in our experiments, and is more computationally efficient.

5 Examples

We tested our new CCD routines in a standard mass-spring cloth simulator with an initially curved sheet of cloth of resolution of 40×400 vertices, dropped on a solid ground plane. As shown in figure 6, this results in a large number of collisions as the cloth stacks up on itself.

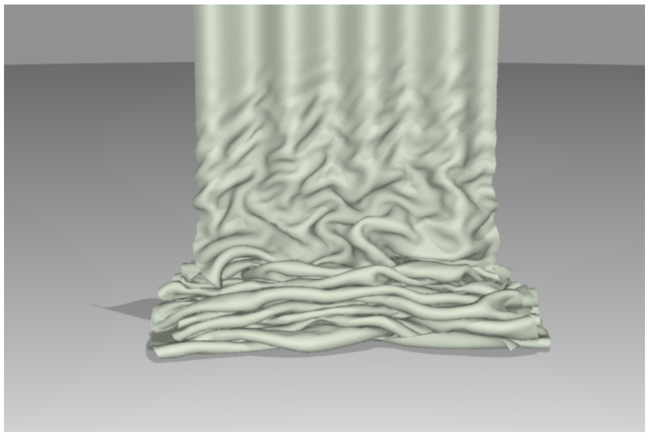

Figure 6: *CCD stress test.*

Although mass-spring systems are popular due to their simplicity and ease of implementation, implementers of cloth animation systems have been turning to increasingly sophisticated models in recent years. For example, the Finite Element Method (FEM) can achieve accurate results and is less dependent on the mesh structure than using edge-based springs [Etzmuss et al. 2003].

In the related sub-field of simulating volumetric elastic solids for graphics, it is becoming increasingly popular to perform on-the-fly optimization of the volumetric simulation mesh [Bargteil et al. 2007; Wojtan and Turk 2008; Wicke et al. 2010]. The benefits of remeshing include reducing error for highly-sheared elements, and concentrating computational effort where it is needed to resolve small-scale details. For cloth, an additional important benefit of remeshing is to increase vertex density in regions of high curvature, so that curved regions can be accurately represented without having to globally refine the surface mesh.

A few authors have suggested refining the simulation elements for cloth [Li and Volkov 2005; Villard and Borouchaki 2005], however, the idea has not been as popular for cloth as it has for solid elasticity. One reason for the lack of uptake is the difficulty in dealing with collisions. For example, adding and removing vertices without introducing self-intersections is a major concern if continuous collision detection assumes that the mesh is intersection-free at the beginning of each time step. Adding and removing vertices and altering the triangulation at discrete times compounds the difficulty in choosing suitable collision and intersection error tolerances.

Armed with our parameterless collision detection system, we demonstrate a complete FEM simulator with on-the-fly, adaptive remeshing. We use linear elasticity with rotated finite elements, as described by Etzmuß et al. [2003], and simple edge crossover springs for bending forces. We choose simple edge splitting, flipping, and collapsing as our mesh optimization operations, and make them all collision safe, using CCD on "pseudo-trajectories", as described by Brochu & Bridson [2009b]. To increase the vertex density in high-curvature areas, we scale the measured edge lengths by local curvature estimates when deciding to collapse or split edges. (We note that these operations are perhaps not as suitable for cloth simulation as a regular subdivision scheme, but it is a reasonable proxy for examining the challenges faced when maintaining intersection-free adaptive surfaces.) Figure 7 shows a frame from a simulation with a single piece of cloth, and the underlying rest-state mesh. We also show a more challenging CCD scenario, with several layers of cloth draped over a solid sphere (figure 1).

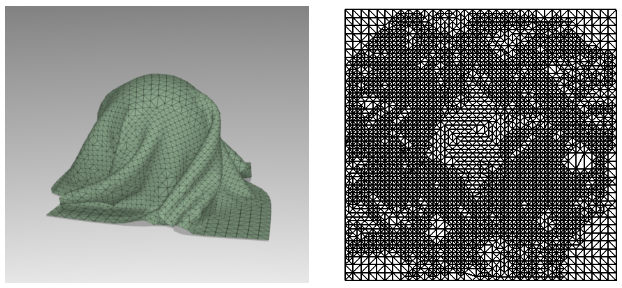

Figure 7: *Cloth with an adaptive simulation mesh*

All of our tests were performed on a single core of a 2.7 GHz Intel i5 processor with 4GB of RAM. We integrated our new CCD algorithm into the open-source El Topo surface tracking library [Brochu and Bridson 2009b], as it provides an intersection-free remeshing framework, suitable for adaptive cloth simulation, and provides an implementation of the cubic-solver based CCD approach for comparison. To test our new CCD, we simply substituted El Topo's inexact CCD and intersection testing functions with our new implementation.

The average time spent per call to CCD, not including culling based on AABB comparisons, but including plane-based culling, was approximately 614 nanoseconds for segment-segment testing, and 439 ns for point-triangle testing. By comparison, the average time in El Topo's cubic solver CCD implementation (again not including AABB culling) was 649 ns and 659 ns.

Counting only tests which were positive (exercising the entire path of our code), the average time per call was 19 microseconds for segment-segment, and 15 µs for point-triangle, compared to 1.2 µs for both tests with the cubic solver CCD. This indicates that without culling, our new algorithm is more expensive than the cubic-solver version, as expected, but also that our culling is very effective.

6 Conclusions

We have presented a novel approach to continuous collision detection, constructed from a set of predicates which can be evaluated exactly. We have shown that the collision detection problem can be rephrased as determining whether a function has an odd number of roots in a given domain. We then showed how this problem can be reduced to testing a ray from the origin against the image of the domain boundary and counting the parity of the crossings. This in turn reduces to a set of ray-vs-triangle and ray-vs-bilinear-patch tests, built from determinant and ϕ sign evaluations.

Our implementation uses a floating-point filter approach for efficiency — first determining if the correct Boolean result can be determined using interval arithmetic, and only using floating-point expansions for exact evaluation if required. We demonstrated the utility of our approach with a challenging test case: simulation of cloth undergoing on-the-fly remeshing with a large amount of contact. To our knowledge, this is the first time an adaptive cloth simulation scheme has been presented which explicitly deals with the challenges of continuous collision detection.

There are several avenues of future work. While irrelevant for the simulations we presented, being able to distinguish zero from a positive even number of roots could be critical for other applications. Our approach should extend naturally from multi-affine functions to testing intersections with higher-degree polynomial patches. Rigid body motion is more naturally expressed in terms of screw motions; though the intermediate positions involve trigonometric functions of time, reparametrization similar to the NURBS approach to conic sections would lead to a multivariate polynomial problem amenable to this attack.

7 Acknowledgments

This work was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada. We would also like to thank Eitan Grinspun and our anonymous reviewers for helpful discussion and comments.

References

BARGTEIL, A. W., WOJTAN, C., HODGINS, J. K., AND TURK, G. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. (Proc. SIGGRAPH) 21*, 3, 594–603.

BROCHU, T., AND BRIDSON, R. 2009. Numerically robust continuous collision detection for dynamic explicit surfaces. Tech. Rep. TR-2009-03, University of British Columbia.

BROCHU, T., AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput. 31*, 4, 2472–2493.

BRÖNNIMANN, H., BURNIKEL, C., AND PION, S. 2001. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics 109*, 25–47.

CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

DEKKER, T. J. 1971. A floating-point technique for extending the available precision. *Numerische Mathematik 18*, 224–242. 10.1007/BF01397083.

ERICSON, C. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

ETZMUSS, O., KECKEISEN, M., AND STRASSER, W. 2003. A fast finite element solution for cloth modelling. In *Proc. 11th Pacific Conference on Computer Graphics and Applications*, 244 – 251.

GOUALARD, F. 2010. Fast and correct SIMD algorithms for interval arithmetic. In *Proceedings of PARA '08*, Springer, Trondheim, Lecture Notes in Computer Science. 10 pages.

GUIGUE, P., AND DEVILLERS, O. 2003. Fast ray-triangle intersection test using orientation predicates. *Journal of Graphics Tools 8*, 1, addendum.

HARMON, D., VOUGA, E., TAMSTORF, R., AND GRINSPUN, E. 2008. Robust treatment of simultaneous collisions. *ACM Trans. Graph. (Proc. SIGGRAPH) 27*, 3, 1–4.

LAMBOV, B. 2006. Interval arithmetic using SSE-2. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, P. Hertling, C. M. Hoffmann, W. Luther, and N. Revol, Eds., vol. 5045 of *Lecture Notes in Computer Science*, 102–113.

LI, L., AND VOLKOV, V. 2005. Cloth animation with adaptively refined meshes. In *Proc. 28th Australasian Conference on Computer Science - Volume 38*, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, ACSC '05, 107–113.

O'REGAN, D., JE CHO, Y., AND CHEN, Y.-Q. 2006. *Topological Degree Theory and Applications*. Chapman and Hall/CRC.

PRIEST, D. M. 1991. Algorithms for arbitrary precision floating point arithmetic. In *Proceedings of the 10th Symposium on Computer Arithmetic*, IEEE Computer Society Press, 132–145.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, 177–89.

RAMSEY, S. D., POTTER, K., AND HANSEN, C. 2004. Ray bilinear patch intersections. *Journal of Graphics Tools 9*, 3, 41–47.

SELLE, A., LENTINE, M., AND FEDKIW, R. 2008. A mass spring model for hair simulation. *ACM Trans. Graph. (Proc. SIGGRAPH) 27*, 64:1–64:11.

SHEWCHUK, J. R. 1996. Robust adaptive floating-point geometric predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, Association for Computing Machinery, 141–150.

STAM, J. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *IEEE CADCG*, 1–11.

TANG, M., KIM, Y. J., AND MANOCHA, D. 2010. Continuous collision detection for non-rigid contact computations using local advancement. *Proc. Int'l. Conf. Robotics and Automation*.

TANG, M., MANOCHA, D., AND TONG, R. 2010. Fast continuous collision detection using deforming non-penetration filters. In *Proc. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games*, 7–13.

VILLARD, J., AND BOROUCHAKI, H. 2005. Adaptive meshing for cloth animation. *Eng. with Comput. 20* (August), 333–341.

WICKE, M., RITCHIE, D., KLINGNER, B., BURKE, S., SHEWCHUK, J., AND O'BRIEN, J. 2010. Dynamic local remeshing for elastoplastic simulation. *ACM Trans. Graphics (Proc. SIGGRAPH) 29*, 3.

WOJTAN, C., AND TURK, G. 2008. Fast viscoelastic behavior with thin features. In *ACM Trans. Graphics (Proc. SIGGRAPH)*, 47.

YAP, C. 2004. Robust geometric computation. In *CRC Handbook of Computational and Discrete Geometry*, J. E. Goodman and J. O'Rourke, Eds., 2 ed. CRC Press LLC, ch. 41.

ZHANG, X., REDON, S., LEE, M., AND KIM, Y. J. 2007. Continuous collision detection for articulated models using Taylor models and temporal culling. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3, 15.

A Implicit Function for a Bilinear Patch

We define the following multivariate polynomial $\phi(\mathbf{x})$ where the indices 0 to 3 refer to the patch corner vertices:

$$\phi(\mathbf{x}) = h_{12}(\mathbf{x}) - h_{03}(\mathbf{x}).$$

The two h-functions are designed to be zero on the straight edges of the patch via products of plane g-functions for the various subsets of three vertices:

$$\begin{aligned} h_{12}(\mathbf{x}) &= g_{012}(\mathbf{x})\, g_{132}(\mathbf{x}) \\ h_{03}(\mathbf{x}) &= g_{013}(\mathbf{x})\, g_{032}(\mathbf{x}) \\ g_{pqr}(\mathbf{x}) &= (\mathbf{x} - \mathbf{x}_p) \cdot (\mathbf{x}_q - \mathbf{x}_p) \times (\mathbf{x_r} - \mathbf{x}_p). \end{aligned}$$

We claim that the zero level set of $\phi(\mathbf{x})$ contains the bilinear patch. To confirm this is indeed the function we seek, take an arbitrary point $\mathbf{x}$ with barycentric coordinates α, β, γ, and δ w.r.t. the corners of the patch:

$$\mathbf{x} = \alpha\mathbf{x}_0 + \beta\mathbf{x}_1 + \gamma\mathbf{x}_2 + \delta\mathbf{x}_3.$$

Recall that the barycentric coordinates of a point with respect to a tetrahedron are proportional to the signed volumes of the tetrahedra formed by the point and each of the triangular faces. Letting V be six times the signed volume of the tetrahedron spanning the corners of the patch, observe that:

$$\begin{aligned} g_{132}(\mathbf{x}) &= \alpha V & \quad g_{032}(\mathbf{x}) &= \beta V \\ g_{013}(\mathbf{x}) &= \gamma V & \quad g_{012}(\mathbf{x}) &= \delta V \end{aligned}$$

Therefore our function evaluates to:

$$\phi(\mathbf{x}) = \delta\alpha V^2 - \gamma\beta V^2.$$

Assuming $V \neq 0$, this is zero if and only if $\alpha\delta = \beta\gamma$, which occurs precisely for the parameterized bilinear surface:

$$\begin{aligned} \alpha &= (1-s)(1-t) & \quad \beta &= (1-s)t \\ \gamma &= s(1-t) & \quad \delta &= st. \end{aligned}$$

Moreover, it is clear that $\phi(\mathbf{x})$ changes sign across the zero level set, dividing space into positive and negative regions separated by the conic containing the bilinear patch.

This construction breaks down if $V = 0$. However this is the case when the patch is perfectly flat, where we can simply replace the entire ray-patch intersection test with two ray-triangle tests.

Supplemental Material to "Efficient Geometrically Exact Continuous Collision Detection"

Tyson Brochu[*]
University of British Columbia

Essex Edwards[*]
University of British Columbia

Robert Bridson[*]
University of British Columbia

1 The Root Parity Lemma and Proof of Correctness

1.1 Outline

We present the root parity lemma to relate the roots of a function to the action of that function on the domain boundary. First, we prove the analog of the lemma for piecewise linear functions defined on a simplicial mesh. Second, we show for any well-behaved C^2 function there exists a piecewise linear interpolant which approximates it sufficiently well to use the first result to prove the root parity lemma. We demonstrate that the intersection function used in collision detection falls into this class of well-behaved C^2 functions, so we argue the correctness of our algorithm on top of the lemma.

1.2 Piecewise Linear Functions

Before proving the root parity lemma, we prove a related result for piecewise linear functions.

Root Parity for Piecewise Linear Functions. *Suppose $\Omega \subset \mathbb{R}^n$ has a simplicial decomposition by mesh M.*

Suppose $\mathbf{F} : \Omega \mapsto \mathbb{R}^n$ is continuous, linear within each simplex of M, has $p < \infty$ roots in Ω, has no roots on $\Gamma = \partial\Omega$, and is one-to-one in a sufficiently small ball around each root.

[*](tbrochu|essex|rbridson)@cs.ubc.ca

Suppose R is a ray from $\mathbf{0}$ to infinity. Call any point $\mathbf{x} \in \Gamma$ such that $\mathbf{F}(\mathbf{x}) \in R$ a crossing point*, then the* crossing number *q is the number of crossing points. Suppose that $\mathbf{F}(\Gamma)$ is smooth at the image of any crossing points, that the ray is not tangent to $\mathbf{F}(\Gamma)$ at any these points, and that $q < \infty$.*

Then, $p \equiv q \bmod 2$.

Suppose the hypotheses are true. Then the image under $\mathbf{F}$ of any simplex in this mesh is also a simplex. So, the image of the entire mesh is also a simplicial mesh, M', though it may be self-intersecting. The function $\mathbf{F}$ is uniquely defined by the position of the vertices of M', and roots of $\mathbf{F}$ are given by simplices in M' that contain the origin.

We may make various simplifying assumptions about M'. Such an assumption will be without loss of generality if, for any mesh M', a perturbation exists that modifies it to satisfy these conditions without changing the parity of the number of roots or the parity of the crossing number for the ray.

Assume the origin does not lie on any facets of M'. This is possible because $\mathbf{F}$ is invertible around each root. Therefore a small perturbation exists which will push the origin from being on a facet to being in only one of the adjacent simplices; therefore not changing the number of roots.

Assume also that M' has no degenerate simplices. That is, each simplex has volume. Simulation of Simplicity [2] addresses almost exactly this problem. A similar argument applies here. The only additional concern is that we do not modify the number of roots. Let $\epsilon > 0$ be the distance from the origin to the nearest facet. Since the perturbation can be arbitrarily small, no vertex needs to be moved more than ϵ, so the number of roots does not change.

Now, consider a ray R from $\mathbf{0}$ to infinity satisfying the conditions of the root parity lemma. Such a ray exists because $\mathbf{F}(\Gamma)$ is smooth almost everywhere and does not include $\mathbf{0}$. Define a *hit* to be an intesection of the ray with the boundary of a simplex. If the ray intersects a facet shared by two simplices, then this is two hits. Each simplex can contribute hits.

First, consider a simplex from M with no roots in it. The image of this simplex does not contain the origin. The ray crosses its boundary either zero or two times. This simplex contributes an even number of hits.

Second, consider a simplex from M containing a root. By hypothesis, the image of this simplex contains the origin in its interior. Consequently, the ray intersects its boundary once, contributing an odd number of hits.

Summing up all the hits, only simplices with roots contribute odd parity to the sum, so the parity of the number of hits equals the parity of the number of roots. Likewise, the parity of the number of hits equals the parity of the crossing number. This is because any intersection of the ray with an interior facet contributes two hits. Only the boundary facets, coincident with Γ, contribute odd parity. So the parity of the number of roots equals the parity of the number of crossings, as was to be shown.

1.3 The Root Parity Lemma

Root Parity Lemma. *Suppose $\Omega \subset \mathbb{R}^n$ is an n-polytope.*

Suppose $\mathbf{F} : \Omega \mapsto \mathbb{R}^n$ is C^2, has $p < \infty$ roots in Ω, has no roots on $\Gamma = \partial\Omega$, and has non-singular Jacobian at each root.

Suppose R is a ray from $\mathbf{0}$ to infinity. Call any point $\mathbf{x} \in \Gamma$ such that $\mathbf{F}(\mathbf{x}) \in R$ a crossing point*, then the* crossing number *q is the number of crossing points. Suppose that $\mathbf{F}(\Gamma)$ is smooth at the image of any crossing points, that the ray is not tangent to $\mathbf{F}(\Gamma)$ at any these points, and that $q < \infty$.*

Then, $p \equiv q \bmod 2$.

Suppose the hypotheses of the root parity lemma are true. Then, let the entire domain Ω be tessellated with a simplicial mesh M, as is possible for a polytope. Let each simplex have circumradius less than δ_{out}, and let each root of $\mathbf{F}$ be at the centroid of a regular simplex. We take the existence of such a mesh to be trivial. Let $\bar{\mathbf{F}}$ be the piecewise linear interpolant of $\mathbf{F}$ on a the mesh M. We argue that there exists such a mesh for which $\bar{\mathbf{F}}$ and $\mathbf{F}$ have the same number of roots, the same crossing number, and $\mathbf{F}$ satisfies the hypotheses of the previous section. From this it follows that the root parity lemma is true.

1.3.1 Roots

In this section, we show that if the mesh is sufficiently fine, then in any simplex, either $\mathbf{F}$ and $\bar{\mathbf{F}}$ both have roots, or neither $\mathbf{F}$ or $\bar{\mathbf{F}}$ have roots.

Let $\mathbf{x}^\star$, be an arbitrary root of $\mathbf{F}$, and let Σ be the simplex containing it. Since this simplex is regular, it has inradius $\delta_{\text{out}} = \kappa\delta_{\text{in}}$ with constant κ. In addition to the functions $\mathbf{F}$ and $\bar{\mathbf{F}}$, we introduce $\hat{\mathbf{F}} = J(\mathbf{x} - \mathbf{x}_i)$ which is the linear approximation to $\mathbf{F}$ about the root (i.e. $J = \nabla\mathbf{F}(\mathbf{x}^\star)$).

Clearly $\hat{\mathbf{F}}(\mathbf{x}^\star) = \mathbf{0}$, so $\hat{\mathbf{F}}$ has a root in Σ. Now, consider a point $\mathbf{q}$ on the surface of Σ.

$$\begin{aligned}\|\hat{\mathbf{F}}(\mathbf{q})\|_2 &= \|J(\mathbf{x}-\mathbf{x}^\star)\|_2\\ &\geq \|J^{-1}\|_2^{-1}\|\mathbf{x}-\mathbf{x}^\star\|_2\\ &\geq \delta_{\text{in}}\|J^{-1}\|_2^{-1}\end{aligned}$$

So, the image of Σ under $\hat{\mathbf{F}}$, which is also a simplex, has its surface at least $\delta_{\text{in}}\|J_i^{-1}\|_2^{-1}$ away from the origin.

By Taylor's Theorem, we have,

$$\|\mathbf{F}(\mathbf{q})-\hat{\mathbf{F}}(\mathbf{q})\| \leq c_1\|\mathbf{q}-\mathbf{q}_i\|^2 \leq c_1\delta_{\text{out}}^2$$

where $c_1 < \infty$ is a constant related to c_t. Similarly, by the approximation quality of linear interpolants (for proof, see e.g. [4]), we have

$$\|\mathbf{F}(\mathbf{q})-\bar{\mathbf{F}}(\mathbf{q})\| \leq c_2\delta_{\text{out}}^2$$

where $c_2 < \infty$ is another constant related to c_t. These can be combined to get the relationship,

$$\|\hat{\mathbf{F}}(\mathbf{q})-\bar{\mathbf{F}}(\mathbf{q})\| \leq c_3\delta_{\text{out}}^2.$$

The origin is at least $\delta_{\text{in}}\|J_i^{-1}\|_2^{-1}$ away from the surface of $\hat{\mathbf{F}}(\Sigma)$, and $\hat{\mathbf{F}}(\Sigma)$ is no more than $c_3\delta_{\text{out}}^2$ away from $\bar{\mathbf{F}}(\Sigma)$. Since $\bar{\mathbf{F}}(\Sigma)$ is also a simplex, it follows that if

$$\delta_{\text{in}}\|J_i^{-1}\|_2^{-1} > c_3{\delta_{\text{out}}}^2 \Leftrightarrow 1/(\kappa c_3\|J^{-1}\|_2) > \delta_{\text{out}},$$

then $\bar{\mathbf{F}}(\Sigma)$ will also contain the origin, and $\bar{\mathbf{F}}$ will also have a root in Σ. Since $\kappa c_3\|J^{-1}\|_2$ is some constant bounded away from zero, we can choose such a δ_{out}. Consequently, both $\mathbf{F}$ and $\bar{\mathbf{F}}$ have a single root in Σ, and $\hat{\mathbf{F}}$ is locally invertible there. Since there are finitely many roots, this constraint on δ_{out} can be met at all roots by some constant $\delta_{\text{out}} > 0$.

Now we show that in the other simplices of the mesh, $\bar{\mathbf{F}}$ has no roots. Continuing with the point $\mathbf{q}$ on the surface of the simplex around a root, we find that

$$\begin{aligned}\|\mathbf{F}(\mathbf{q})\| &\geq \|\hat{\mathbf{F}}(\mathbf{q})\| - c_1\|\mathbf{q}-\mathbf{x}\|^2\\ &\geq \|J(\mathbf{q}-\mathbf{x})\| - c_1{\delta_{\text{out}}}^2\\ &\geq \|J^{-1}\|^{-1}\|\mathbf{q}-\mathbf{x}\| - c_1{\delta_{\text{out}}}^2\\ &\geq \|J^{-1}\|^{-1}\delta_{\text{in}} - c_1{\delta_{\text{out}}}^2\\ &\geq \|J^{-1}\|^{-1}\delta_{\text{in}} - c_1\kappa^2{\delta_{\text{in}}}^2\\ &\geq \|J^{-1}\|^{-1}\kappa^{-1}\delta_{\text{out}} - c_1{\delta_{\text{out}}}^2\\ &\geq \alpha\delta_{\text{out}} - c_1{\delta_{\text{out}}}^2\end{aligned}$$

where $\alpha > 0$ is the minimum value over all roots of $\|J^{-1}\|^{-1}\kappa^{-1}$.

Let S be the union of the simplices which contain roots of $\mathbf{F}$, as have already been addressed. Then, $\Omega_0 = \overline{\Omega \setminus S}$ is the remainder of the domain, including the boundaries. In Ω_0, $\|\mathbf{F}\| > F_{\text{min}} > 0$. For sufficiently small simplices, the minimum value of $\|\mathbf{F}\|$ will occur on the surface of one of the simplices surrounding a root. For simplicity, assume that this is the case. So, $F_{\text{min}} > \alpha\delta_{\text{out}} - c_1{\delta_{\text{out}}}^2$ everywhere outside the simplices surrounding the roots. Then at all points $\mathbf{x}$ outside the root-simplices $\|\bar{\mathbf{F}}(\mathbf{x}) - \mathbf{F}(\mathbf{x})\| \leq c_2\delta_{\text{out}}^2$, and so

$$\begin{aligned}
\|\bar{\mathbf{F}}(\mathbf{x})\| &\geq \|\mathbf{F}(\mathbf{x})\| - c_2\delta_{\text{out}}^2 \\
&\geq F_{\text{min}} - c_2\delta_{\text{out}}^2 \\
&> \alpha\delta_{\text{out}} - c_1{\delta_{\text{out}}}^2 - c_2\delta_{\text{out}}^2 \\
&> \alpha\delta_{\text{out}} - {\delta_{\text{out}}}^2(c_1 + c_2)
\end{aligned}$$

For sufficiently small δ_{out}, we have $\|\bar{\mathbf{F}}(\mathbf{x})\| > 0$ because $\alpha > 0$. So, outside of the simplices surrounding the roots, $\bar{\mathbf{F}}$ has no roots.

It follows that $\mathbf{F}(x)$ and $\bar{\mathbf{F}}(x)$ have the same number of roots.

1.3.2 Crossing Points

To show that $\bar{\mathbf{F}}$ also has the same crossing number as $\mathbf{F}(\Gamma)$, we construct two new functions $H : \Gamma \mapsto \mathbb{R}$ and $\mathbf{G} : \Gamma \mapsto \mathbb{R}^{n-1}$. Without loss of generality, by a simple rotation, let the ray be the positive x_1 axis. Then, let $H(\mathbf{x}) = \mathbf{F}_1(x)$ be the first component of $\mathbf{F}(\mathbf{x})$. It measures the distance along the ray of the closest point to $\mathbf{F}(\mathbf{x})$. Second, let $\mathbf{G}(\mathbf{x})$ be components 2 through n of $\mathbf{F}(\mathbf{x})$. So, it measures a vector-distance from $\mathbf{F}(\mathbf{x})$ to the closest point on the ray. Consequently, a point $\mathbf{x}$ is a crossing point of R and $\mathbf{F}(\Gamma)$ iff $H(\mathbf{x}) > 0$ and $\mathbf{G}(\mathbf{x}) = \mathbf{0}$.

Now, consider the functions $\bar{\mathbf{G}}$ and $\bar{H}$ defined as above, but using $\bar{\mathbf{F}}$ instead of $\mathbf{F}$. As before, $\mathbf{x}$ is a crossing point of R and $\bar{\mathbf{F}}(\Gamma)$ iff $\bar{H}(x) > 0$ and $\bar{\mathbf{G}}(x) = \mathbf{0}$. Notice also, $\bar{H}$ and $\bar{\mathbf{G}}$ are the linear interpolants of H and $\mathbf{G}$ on the boundary elements of the mesh, which form an $n-1$ dimensional simplex mesh in Γ. So we can use similar arguments as above to establish an exact correspondence between the crossing points of R and $\mathbf{F}(\Gamma)$ and the crossing points of R and $\bar{\mathbf{F}}(\Gamma)$ by matching the roots of $\mathbf{G}$ and $\bar{\mathbf{G}}$ and the signs of H and $\bar{H}$ at those roots.

$\mathbf{G}$ may not be locally invertible at all of its roots, so we take a different approach than above for $\mathbf{F}$. Add all of the crossing points of R and $\mathbf{F}(\Gamma)$ as vertices to the mesh, by hypothesis there are a finite number of them. This does not contradict the earlier construction of a single simplex around each root of $\mathbf{F}$, because $\mathbf{F}$ has no roots on Γ. With these vertices in the mesh, $H(\mathbf{x}) > 0$ and $\mathbf{G}(\mathbf{x}) = \mathbf{0}$ implies $\bar{\mathbf{G}}(\mathbf{x}) = \mathbf{0}$. Let any simplex containing a root of $\mathbf{G}$ be small enough that it contains only one. By construction, it will be at a vertex, the *root-vertex*. All the other vertices of this simplex form an $(n-2)$-face, the *far-face*. By using anisotropic simplices around the roots, the far-face can always be distance $\delta > 0$ from the root, but fit in an arbitrarily small ball of radius r. Consequently, $\|\mathbf{G}\| > \alpha > 0$ on the far-face. A described earlier, with a sufficiently fine mesh, $\bar{G}$ will have no roots on the far-face, and consequently no roots anywhere in the simplex, except at the root-vertex.

Let ϵ be half the minimum magnitude of H at any root of $\mathbf{G}$. When $-\epsilon < H < \epsilon$, then $\|\mathbf{G}\| > \delta > 0$. So, in a sufficiently fine mesh, $\bar{\mathbf{G}} \neq \mathbf{0}$. This follows from the same bound used above to analyze $\bar{\mathbf{F}}$. When $H < -\epsilon$, in a sufficiently fine mesh we get $\bar{H} < 0$. Finally, when $H > \epsilon$, a sufficiently fine mesh will have $\bar{H} > 0$ and, outside of the simplices constructed above, $\bar{\mathbf{G}} \neq \mathbf{0}$. Combined with the results above, we conclude that $\mathbf{F}$ and $\bar{\mathbf{F}}$ have the same crossing points.

Combined with the earlier result, we have that $\mathbf{F}$ and $\bar{\mathbf{F}}$ have the same number of roots, and exactly the same crossing points. So we have shown that the root parity lemma is true.

1.4 Toplogical Degree

The root parity lemma is closely related to proofs and concepts from topological degree theory. In some sense, it is a specialization, and our proof is subsequently tailored for our application.

Topological degree defines a multidimensional generalization of the winding number. One definition of topological degree is the sum of the signs of the Jacobian determinants at all the roots of $\mathbf{F}$. For non-singular roots, the parity of topological degree is the same as the parity of the number of roots, which is what we want to measure. We refer the interested reader to the text by O'Regan et. al [3] for more details about topological degree.

While the definition above is in terms of the roots in the domain's interior, an equivalent expression depends only on the boundary. In fact, it can be calculated by summing ± 1 (depending on some property of $\mathbf{F}$ and its derivatives) at each intersection between a ray and the image of the boundary. Such an approach is described, for example, by Aberth in his text on numerical methods [1]. Because we only care about parity, our algorithm can simply count the number of intersections.

1.5 Proof of the Collision Algorithm

The function that the algorithm uses is C^2 with bounded curvature, defined in a polytope domain. When the input geometry to the collision detection algorithm produces a function $\mathbf{F}$ satisfying the hypotheses of the root parity lemma, then the correctness of the algorithm follows trivially.

However, $\mathbf{F}$ doesn't always satisfy the hypotheses. It is always C^2 and it always has bounded second derivative. However, it may have roots on Γ, it may have infinitely many roots, and it may have a singular Jacobian at any root. We argue here that the algorithm still does the right thing for collision detection in this case.

If $\mathbf{F}(\mathbf{x}) = 0$ on Γ, then $\mathbf{F}(\Gamma)$ is at the origin. So, the ray's vertex lies on the surface. The algorithm immediately identifies this as a collision without bothering with ray crossings.

If $\mathbf{F}$ has an infinite number of roots, then because of the multi-affine form of $\mathbf{F}$, it must have a root on Γ, and this case is detected as above.

Otherwise, if $\nabla\mathbf{F}$ is not invertible at a root, then there exists a small perturbation to $\mathbf{F}$ strictly in the interior of the domain, such that it is not singular at any roots and is unchanged on Γ. This perturbation does not affect the initial and final configuration of the mesh or the linear trajectories of the extremes of the mesh elements: it only causes an arbitrarily small adjustment to the interpolated trajectories. Thus, it will not change whether or not a significant collision has occurred, and the algorithm returns the correct result.

The root parity lemma also has conditions on the ray that the algorithm must meet. The only non-smooth regions of Γ are the edges. The algorithm traces a new ray when an intersection with an edge is detected. If the crossing number is infinite, then the ray must hit the edges, which is detected as above. The requirement that the ray not be tangent is handled by the ray-patch parity algorithm, which returns the correct (even) parity in that case.

We note that while this proof contains several cases involving perturbing $\mathbf{F}$ or the ray, the algorithm does not need to do this. This is only a conceptual perturbation for use in the proof.

2 Discussion of the odd-parity counting argument

In this section, we discuss the correctness of ignoring an even number of collisions in the CCD algorithm outlined in the main paper. The claim made in the paper is that if a pair of mesh elements (a triangle and point, or two edges) whose vertices are moving on constant speed, linear trajectories collide an even number of times over the course of a time step, we can safely ignore the collisions (i.e. no collision resolution is required).

Although we do not have a rigorous proof for the correctness of this approach, we argue here that at the very least, a few desirable properties can be maintained. This model also will also clearly result in different behaviour than detecting and resolving every collision, however most collision processing systems already make simplifying assumptions such as not solving collisions in order of simulation time, resulting in an approximate yet plausible solution.

2.1 No self-intersections are introduced

Suppose we are given a mesh that is intersection-free at $t = 0$, and has at least one edge-triangle intersection at $t = 1$. Setting aside collisions involving time or geometry boundaries (which are always flagged by our algorithm), this implies there must have been an odd number of total collisions between the edge and triangle sub-elements (i.e. between the two edge end points and the triangle, and between the three triangle edges and the intersecting edge). This follows from the fact that any such collision changes the edge-triangle pair from an intersecting state to a non-intersecting state, or vice-versa.

2.2 Disjoint volumes defined by closed mesh surfaces remain disjoint

Suppose we have two closed meshes which define disjoint volumes (i.e. one mesh does not intersect or lie inside the other) at $t = 0$. Then at $t = 1$, if one mesh lies inside the other, our CCD algorithm would report a collision. A vertex outside a closed surface must collide with the surface an odd number of times to end up inside the surface. If there is an odd number of collisions between the vertex and the surface as a whole, then there must be at least one triangle with an odd number of collisions against the vertex.

References

[1] O. Aberth. *Precise Numerical Methods Using C++*. Academic Press, 1998.

[2] H. Edelsbrunner and E. P. Mcke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph*, 9:66–104, 1990.

[3] D. O'Regan, Y. Je Cho, and Y.-Q. Chen. *Topological Degree Theory and Applications*. Chapman and Hall/CRC, 2006.

[4] J. R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *In 11th International Meshing Roundtable*, pages 115–126, 2002.